

spe: A first implementation

version 1.0

Overview

This document presents the details necessary to use the first pass at the in-kernel spe.

/etc/policies.spe

This flat file serves as the stable storage for the policies. It is a list of policies where a line can either be a comment (first char is a '#') or a policy. The policies are read in order and sorted by id.

Each policy is of the form:

id, stripe count, unit size, guids, attribute expression

Where the number of guids must be at least the stripe count.

And the attribute expression is a boolean rule composed of the operators &&, ||, and !. These join expressions of the form:

ATTRIBUTE comparison-op VALUE

Where the comparison-op is one of '==' and '!='. The allowed attributes are:

- base -- base part of the file name, without the extension
- day -- day of the month in numeric form
- domain -- DNS subdomain
- ext -- file extension
- file -- base and extension
- fqdn -- fully qualified domain name
- gid -- Group ID
- hour -- Hour of the day
- host -- Short hostname
- ip -- Machine address
- path -- Path including parent directory
- subnet -- Network Address, with a netmask (/24)

- uid -- User ID
- weekday -- day of the week in short form: “mon”

And the values are strings appropriate for the attributes.

Getting guides

Work in progress!!!

Either I provide a tool to extract guides from the MDS or I try to automate this a bit.

I.e., a mdb script or some stand alone program to extract guides from the REPORTAVAIL.

The automation would be to create a file called /etc/npools.spe. It would have DS paths, i.e., ds1:/dspool, and these would be sent to the spe in the kernel along with the policies. The policies would contain lists of npool names instead of lists of 64 bit integers.

The spe code would then be responsible for mining the REPORTAVAIL to get the guides.

The benefits to this approach are that:

1. We don't mess with 64 bit guides.
2. We mimic the original pfsadm schema of npools and policies.

Default action

If the file is not present, then a default layout is generated.

/usr/lib/nfs/sped command

Policies are not loaded into the kernel until the sped command is run. If it is not run, then a default policy is selected.

nfssys() Interface

NFS_SPE is a new nfssys_op for the nfssys() syscall. It uses a struct nfsspe_args

```
struct nfsspe_args {
    nfsspe_op_t    nsa_opcode;    /* operation discriminator */
    uint_t        nsa_did;       /* Door id to upcall */
    char          *nsa_xdr;       /* XDR data */
    size_t        nsa_xdr_len;    /* Size of XDR data */
};
```

Where the opcodes are:

```
typedef enum nfsspe_op {
    SPE_OP_SET_DOOR,          /* Not in Use */
    SPE_OP_POLICY_POPULATE, /* Unload policies and use new ones */
    SPE_OP_POLICY_NUKE,      /* Unload all policies */
    SPE_OP_POLICY_ADD,       /* Add a single policy */
    SPE_OP_POLICY_DELETE,    /* Delete a single policy */
    SPE_OP_SCHEDULE          /* Not in Use */
} nfsspe_op_t;
```

When `spe` loads the policies from `/etc/policies.spe`, it converts them into XDR format for transfer to the kernel.

Kernel Interface

```
int
spe_allocate(vattr_t *vap, struct netbuf *addr, char *dir_path,
             count4 *lh_stripe_count, uint32_t *lh_unit_size,
             uint64_t **guids)
```

- `vap` is a `vattr` and supplies the UID and GID of the creator of the file
- `addr` is the machine address of the client
- `dir_path` is the full path to the file
- `lh_stripe_count` is a pointer to the actual `stripe_count`
- `lh_unit_size` is a pointer to the actual unit size
- `guids` is a pointer to an array of `guids` -- `spe_allocate()` will create the array and the caller is responsible for freeing the memory.